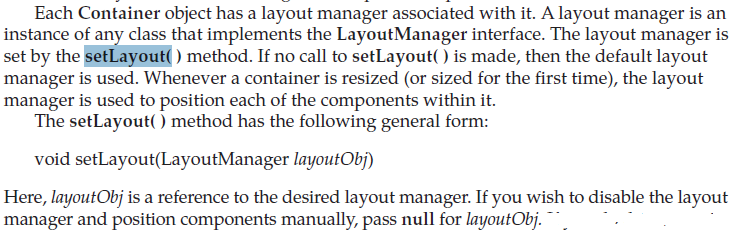
**LayoutManagers**

* **The LayoutManagers are used to arrange components in a particular manner.**
* The **LayoutManagers** facilitates us to control the positioning and size of the components in GUI forms.
* **LayoutManager is an interface that is implemented by all the classes of layout managers.**
* **A layout manager is an object that implements the LayoutManager interface and determines the size and position of the components within a container**. Although components can provide size and alignment hints, a container's layout manager has the final say on the size and position of the components within the container.
* There are the following classes that represent the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

**Note 1: Understanding the SetLayout()**

****

* **Note 2:**
* The setLayout(...) Method allows you to set the layout of the container, often a JPanel, to say FlowLayout, BorderLayout, GridLayout, null layout, or whatever layout desired. The layout manager helps lay out the components held by this container.
* **Note 3 :**
* FlowLayout Manager is the default layout manager of the applet or panel.

**BorderLayout:**

* **The BorderLayout is used to arrange the components in five regions: north, south, east, west, and center.**
* Each region (area) may contain one component only.
* It is the default layout of a frame or window.
* The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

**Constructors of BorderLayout class:**

* **BorderLayout():** creates a border layout but with no gaps between the components.
* **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

**Example of BorderLayout class:**

importjava.awt.\*;

importjavax.swing.\*;

class Border

{

JFrame f;

Border()

{

f = new JFrame();

// creating buttons

JButton b1 = new JButton("NORTH");; // the button will be labeled as NORTH

JButton b2 = new JButton("SOUTH");; // the button will be labeled as SOUTH

JButton b3 = new JButton("EAST");; // the button will be labeled as EAST

JButton b4 = new JButton("WEST");; // the button will be labeled as WEST

JButton b5 = new JButton("CENTER");; // the button will be labeled as CENTER

f.add(b1, BorderLayout.NORTH); // b1 will be placed in the North Direction

f.add(b2, BorderLayout.SOUTH); // b2 will be placed in the South Direction

f.add(b3, BorderLayout.EAST); // b2 will be placed in the East Direction

f.add(b4, BorderLayout.WEST); // b2 will be placed in the West Direction

f.add(b5, BorderLayout.CENTER); // b2 will be placed in the Center

f.setSize(300, 300);

f.setVisible(true);

}

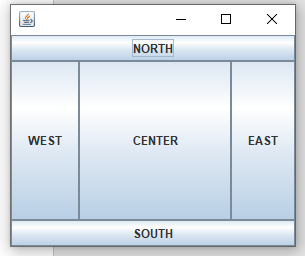
public static void main(String[] args) {

new Border();

}

}

**Output:**



**Example of BorderLayout class: Using BorderLayout(int hgap, int vgap) constructor**

* The following example inserts horizontal and vertical gaps between buttons using the parameterized constructor BorderLayout(inthgap, int gap)

importjava.awt.\*;

importjavax.swing.\*;

public class BorderLayoutExample

{

JFramejframe;

// constructor

BorderLayoutExample()

{

// creating a Frame

jframe = new JFrame();

// create buttons

JButtonbtn1 = new JButton("NORTH");

JButtonbtn2 = new JButton("SOUTH");

JButtonbtn3 = new JButton("EAST");

JButtonbtn4 = new JButton("WEST");

JButtonbtn5 = new JButton("CENTER");

// creating an object of the BorderLayout class using the parameterized constructor where the horizontal gap is 20 and vertical gap is 15. The gap will be evident when buttons are placed in the frame

jframe.setLayout(new BorderLayout(20,15));

jframe.add(btn1, BorderLayout.NORTH);

jframe.add(btn2, BorderLayout.SOUTH);

jframe.add(btn3, BorderLayout.EAST);

jframe.add(btn4, BorderLayout.WEST);

jframe.add(btn5, BorderLayout.CENTER);

jframe.setSize(300,300);

jframe.setVisible(true);

}

// main method

public static void main(String argvs[])

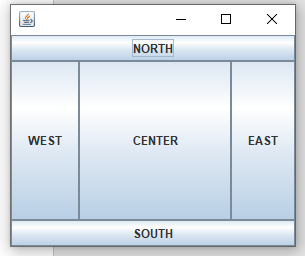
{

newBorderLayoutExample();

}

}

**Output:**



Example 3 : Java BorderLayout: Without Specifying Region

The add() method of the JFrame class can work even when we do not specify the region. In such a case, only the latest component added is shown in the frame, and all the components added previously get discarded. The latest component covers the whole area. The following example shows the same.

importjava.awt.\*;

importjavax.swing.\*;

public class BorderLayoutWithoutRegionExample

{

JFramejframe;

// constructor

BorderLayoutWithoutRegionExample()

{

jframe = new JFrame();

JButtonbtn1 = new JButton("NORTH");

JButtonbtn2 = new JButton("SOUTH");

JButtonbtn3 = new JButton("EAST");

JButtonbtn4 = new JButton("WEST");

JButtonbtn5 = new JButton("CENTER");

// horizontal gap is 7, and the vertical gap is 7

// Since region is not specified, the gaps are of no use

jframe.setLayout(new BorderLayout(7, 7));

// each button covers the whole areahowever, the btn5 is the latest button

// that is added to the frame; therefore, btn5is shown

jframe.add(btn1);

jframe.add(btn2);

jframe.add(btn3);

jframe.add(btn4);

jframe.add(btn5);

jframe.setSize(300,300);

jframe.setVisible(true);

}

// main method

public static void main(String argvs[])

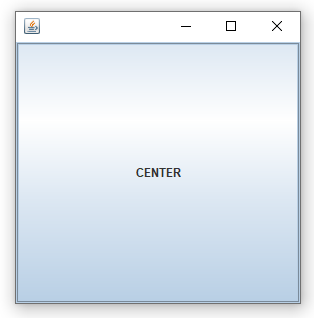
{

newBorderLayoutWithoutRegionExample();

}

}

**Output:**



GridLayout

**The Java GridLayout class is used to arrange the components in a rectangular grid. One component is displayed in each rectangle.**

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

Example of GridLayout class: Using GridLayout() Constructor

The GridLayout() constructor creates only one row. The following example shows the usage of the parameterless constructor.

importjava.awt.\*;

importjavax.swing.\*;

classGridLayoutExample

{

JFrameframeObj;

// constructor

GridLayoutExample()

{

frameObj = new JFrame();

// creating 9 buttons

JButtonbtn1 = new JButton("1");

JButtonbtn2 = new JButton("2");

JButtonbtn3 = new JButton("3");

JButtonbtn4 = new JButton("4");

JButtonbtn5 = new JButton("5");

JButtonbtn6 = new JButton("6");

JButtonbtn7 = new JButton("7");

JButtonbtn8 = new JButton("8");

JButtonbtn9 = new JButton("9");

// adding buttons to the frame

// since, we are using the parameterless constructor, therfore;

// the number of columns is equal to the number of buttons we

// are adding to the frame. The row count remains one.

frameObj.add(btn1); frameObj.add(btn2); frameObj.add(btn3);

frameObj.add(btn4); frameObj.add(btn5); frameObj.add(btn6);

frameObj.add(btn7); frameObj.add(btn8); frameObj.add(btn9);

// setting the grid layout using the parameterless constructor

frameObj.setLayout(new GridLayout());

frameObj.setSize(300, 300);

frameObj.setVisible(true);

}

// main method

public static void main(String argvs[])

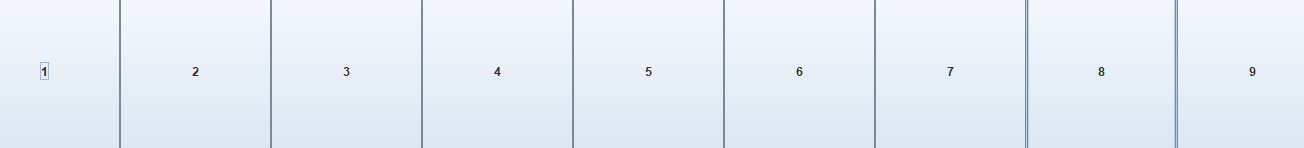
{

newGridLayoutExample();

}

}

**Output:**



Example of GridLayout class: Using GridLayout(int rows, int columns) Constructor:

importjava.awt.\*;

importjavax.swing.\*;

public class MyGridLayout{

JFrame f;

MyGridLayout(){

f=new JFrame();

JButton b1=new JButton("1");

JButton b2=new JButton("2");

JButton b3=new JButton("3");

JButton b4=new JButton("4");

JButton b5=new JButton("5");

JButton b6=new JButton("6");

JButton b7=new JButton("7");

JButton b8=new JButton("8");

JButton b9=new JButton("9");

// adding buttons to the frame

f.add(b1); f.add(b2); f.add(b3);

f.add(b4); f.add(b5); f.add(b6);

f.add(b7); f.add(b8); f.add(b9);

// setting grid layout of 3 rows and 3 columns

f.setLayout(new GridLayout(3,3));

f.setSize(300,300);

f.setVisible(true);

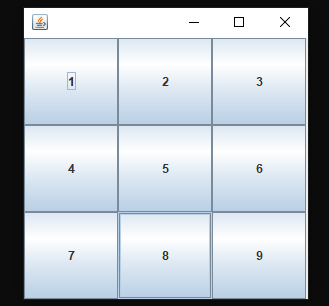
}

public static void main(String[] args) {

newMyGridLayout();

}

}



FlowLayout

* **The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow).**
* **It is the default layout manager of the applet or panel.**

Fields of FlowLayout class

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, inthgap, intvgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example of FlowLayout class: Using FlowLayout() constructor

importjava.awt.\*;

importjavax.swing.\*;

classFlowLayoutExample

{

JFrameframeObj;

// constructor

FlowLayoutExample()

{

// creating a frame object

frameObj = new JFrame();

// creating the buttons

JButton b1 = new JButton("1");

JButton b2 = new JButton("2");

JButton b3 = new JButton("3");

JButton b4 = new JButton("4");

JButton b5 = new JButton("5");

JButton b6 = new JButton("6");

JButton b7 = new JButton("7");

JButton b8 = new JButton("8");

JButton b9 = new JButton("9");

JButton b10 = new JButton("10");

// adding the buttons to frame

frameObj.add(b1); frameObj.add(b2); frameObj.add(b3); frameObj.add(b4);

frameObj.add(b5); frameObj.add(b6); frameObj.add(b7); frameObj.add(b8);

frameObj.add(b9); frameObj.add(b10);

// parameter less constructor is used

// therefore, alignment is center

// horizontal as well as the vertical gap is 5 units.

frameObj.setLayout(new FlowLayout());

frameObj.setSize(300, 300);

frameObj.setVisible(true);

}

// main method

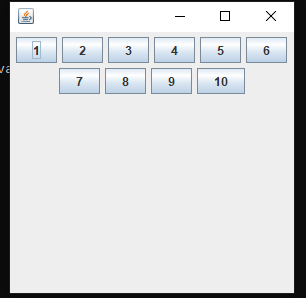
public static void main(String argvs[])

{

newFlowLayoutExample();

}

}



**CardLayout**

**The Java CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.**

Constructors of CardLayout Class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(inthgap, intvgap):** creates a card layout with the given horizontal and vertical gap.

Commonly Used Methods of CardLayout Class

* **public void next(Container parent):** is used to flip to the next card of the given container.
* **public void previous(Container parent):** is used to flip to the previous card of the given container.
* **public void first(Container parent):** is used to flip to the first card of the given container.
* **public void last(Container parent):** is used to flip to the last card of the given container.
* **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

**Example of CardLayout Class: Using Default Constructor**

The following program uses the next() method to move to the next card of the container.

importjava.awt.\*;

importjavax.swing.\*;

importjava.awt.event.\*;

classCardLayoutExample1 extends JFrame implements ActionListener

{

CardLayoutcrd;

// button variables to hold the references of buttons

JButtonbtn1, btn2, btn3;

Container cPane;

// constructor of the class

CardLayoutExample1()

{

cPane = getContentPane();

//default constructor used

// therefore, components will

// cover the whole area

crd = new CardLayout();

cPane.setLayout(crd);

// creating the buttons

btn1 = new JButton("K");

btn2 = new JButton("SIVA");

btn3 = new JButton("KUMAR");

// adding listeners to it

btn1.addActionListener(this);

btn2.addActionListener(this);

btn3.addActionListener(this);

cPane.add("a", btn1); // first card is the button btn1

cPane.add("b", btn2); // first card is the button btn2

cPane.add("c", btn3); // first card is the button btn3

}

public void actionPerformed(ActionEvent e)

{

// Upon clicking the button, the next card of the container is shown

// after the last card, again, the first card of the container is shown upon clicking

crd.next(cPane);

}

// main method

public static void main(String argvs[])

{

// creating an object of the class CardLayoutExample1

CardLayoutExample1crdl = new CardLayoutExample1();

// size is 300 \* 300

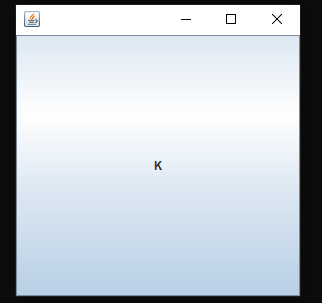
crdl.setSize(300, 300);

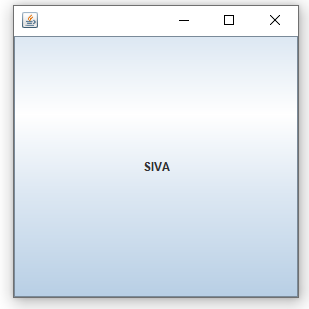
crdl.setVisible(true);

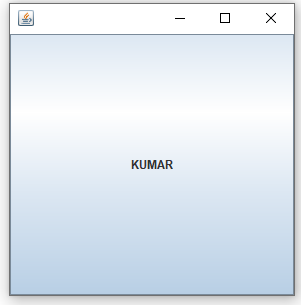
crdl.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}







**GridBagLayout**

**The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.**

The components may not be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of the constraints object, we arrange the component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine the component's size. GridBagLayout components are also arranged in the rectangular grid but can have many different sizes and can occupy multiple rows or columns.

Constructor

**GridBagLayout():** The parameterless constructor is used to create a grid bag layout manager.

importjava.awt.Button;

importjava.awt.GridBagConstraints;

importjava.awt.GridBagLayout;

importjavax.swing.\*;

classGridBagLayoutExample extends JFrame{

public static void main(String[] args) {

GridBagLayoutExample a = new GridBagLayoutExample();

}

publicGridBagLayoutExample() {

GridBagLayout grid = new GridBagLayout();

GridBagConstraintsgbc = new GridBagConstraints();

setLayout(grid);

setTitle("GridBag Layout Example");

GridBagLayout layout = new GridBagLayout();

this.setLayout(layout);

gbc.fill = GridBagConstraints.HORIZONTAL;

gbc.gridx = 0;

gbc.gridy = 0;

this.add(new Button("Button One"), gbc);

gbc.gridx = 1;

gbc.gridy = 0;

this.add(new Button("Button two"), gbc);

gbc.fill = GridBagConstraints.HORIZONTAL;

gbc.ipady = 20;

gbc.gridx = 0;

gbc.gridy = 1;

this.add(new Button("Button Three"), gbc);

gbc.gridx = 1;

gbc.gridy = 1;

this.add(new Button("Button Four"), gbc);

gbc.gridx = 0;

gbc.gridy = 2;

gbc.fill = GridBagConstraints.HORIZONTAL;

gbc.gridwidth = 2;

this.add(new Button("Button Five"), gbc);

setSize(300, 300);

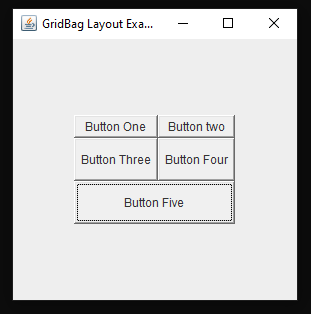
setPreferredSize(getSize());

setVisible(true);

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

}

}



**Note:** A **GridLayout**puts all the components in a rectangular grid and is divided into **equal-sized rectangles** and each component is placed inside a rectangle whereas **GridBagLayout**is a **flexible layout manager**that aligns the components **vertically and horizontally** without requiring that the components be of the same size. Each **GridBagLayout**object maintains a dynamic, rectangular **grid of cells** with each component occupying one or more cells called **Component display area**.